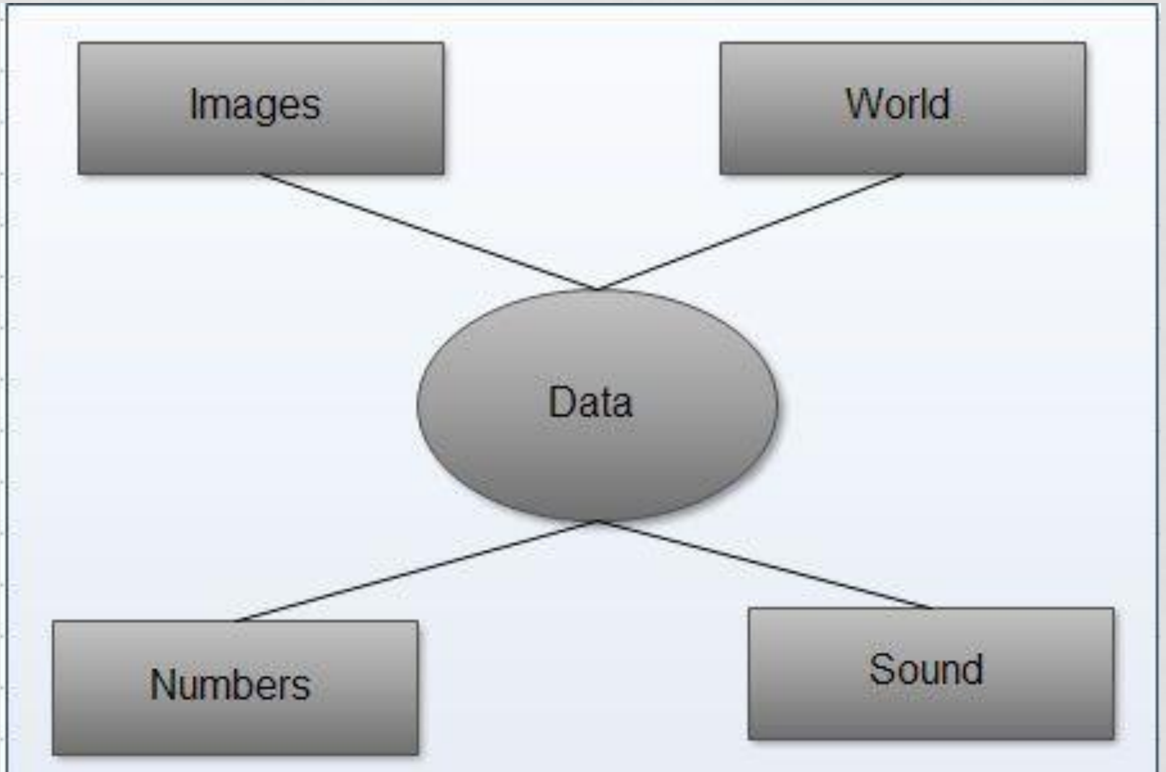
DATABASE MANAGEMENT SYSTEM

UNIT 1

DATA



Different forms of data representation

INFORMATION

- Information is organised or classified data, which has some meaningful values for the receiver.
- Information is data that has been converted into more useful or intelligible form.

Characteristics of Information:

- Timely
- Accuracy
- completeness

Difference between data and information

Data	Information
1. Derived from Latin word 'Datum'	1. Derived from word 'informare'
2. Data is raw fact.	2. Processed form of data.
3. May or may not be meaningful.	3. Always meaningful.
4. Input to any system may be treated as data.	4. Output after processing system is information.
5. Understanding is difficult	5. Understanding is easy.
6. Data may not be in order.	6. Information should be in order.
7. Example: survey data	7. Example: census report

DATABASE

- A database is a collection of related data which represents some aspect of the real world. A database system is designed to be built and populated with data for a certain task

Two main types of database

Flat file database	Relational database
Use on personal computer.	Used a lot in large organisations.
Store data in one file at a time and this must store all the data.	Store data in separate tables and files. (have links/ relationships which allow data from another file to be shown, used and edited but not copy in a current file. So that whenever the values the other file change, the data displayed in the current file also changes.
Not suitable for large database because they can be slow, have large file size and use a lot of memory.	

WHAT IS DBMS?

- Collection of interrelated data
- Set of programs to access the data
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both *convenient* and *efficient* to use.

DBMS

- DBMS is a collection of organized, interrelated data and set of programs to store the data efficiently and access those data in an easy and effective manner

A software package/ system to facilitate the creation and maintenance of a computerized database.

It

- defines (data types, structures, constraints)
- construct (storing data on some storage medium

controlled by DBMS)

- manipulate (querying, update, report generation) databases for various applications.

ADVANTAGES OF DBMS

- Reduction of Redundancies
- Elimination of Inconsistencies
- Shared Data
- Integrity
- Security
- Data Independence
  - Physical Data Independence
  - Logical Data Independence

DISADVANTAGES OF DBMS

- Backup and Recovery Issues
- Complexity
- Performance
- Security
- Problem associated with centralization

NEED OF DBMS

- Creation of a database.
- Retrieval of information from the database.
- Updating the database.
- Managing a database.
- Storing Database

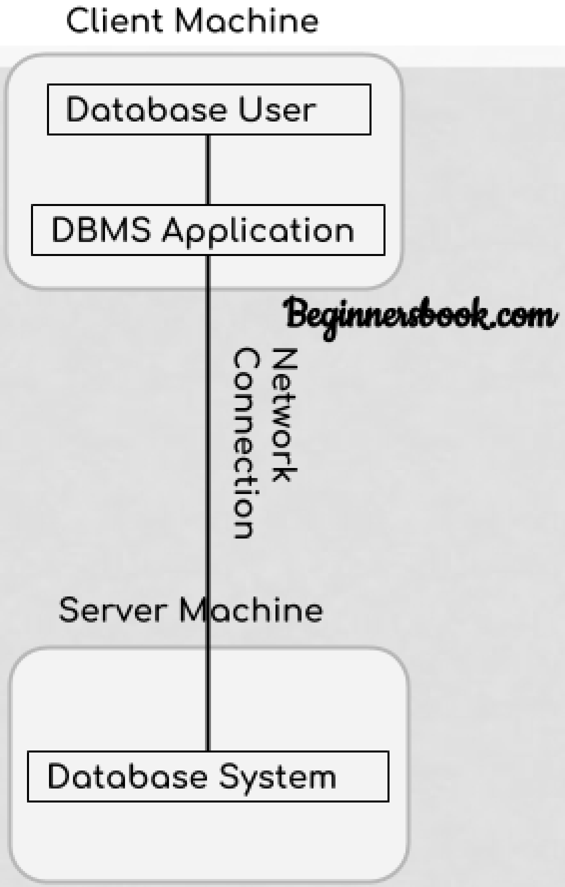
APPLICATIONS OF DBMS

- Home
- Banking
- Reservation System
- Finance
- E-Commerce
- Industry
- Education
- Sales

ARCHITECTURE OF DBMS

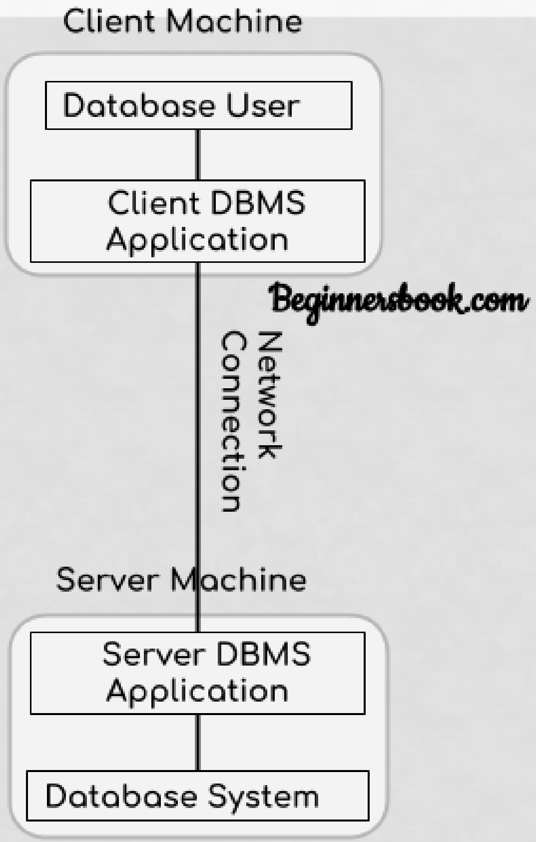
- **1. Single tier architecture**
- In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.
- For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

TWO TIER ARCHITECTURE



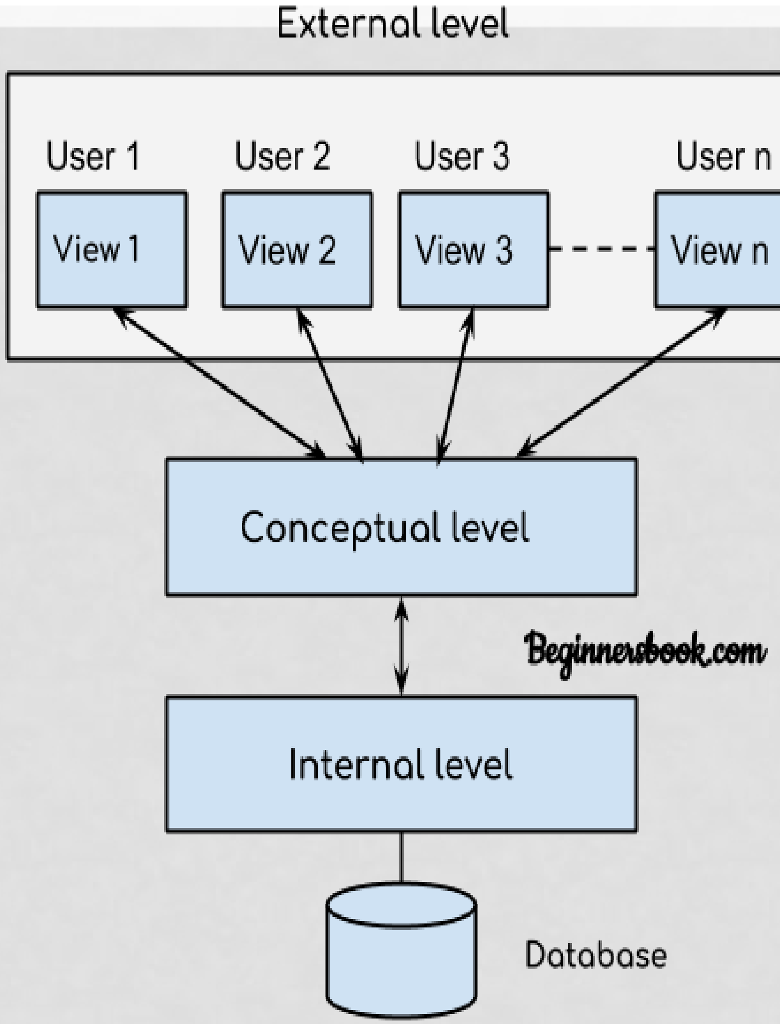
Two-Tier architecture

THREE TIER ARCHITECTURE



Three-Tier architecture

THREE LEVEL ARCHITECTURE OF DATABASE

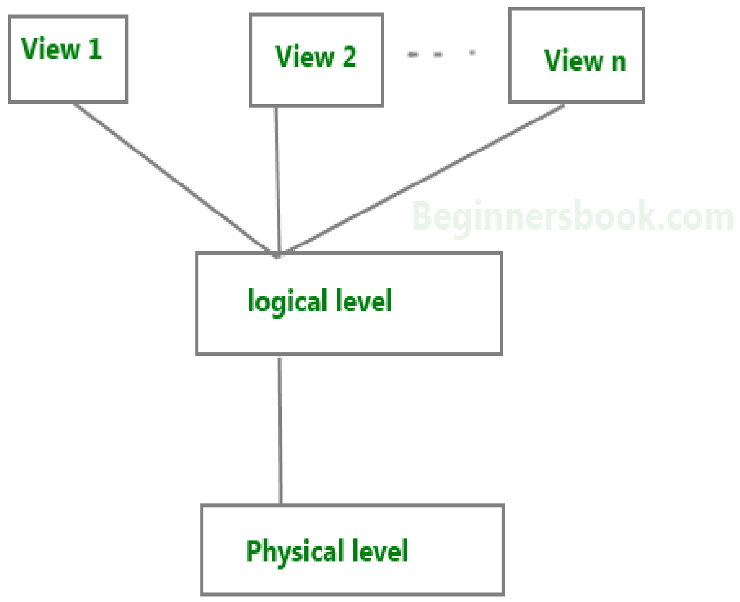


Compiled by Ms. Prajakta Joshi

VIEW OF DATA

- Data abstraction
- Instance and schema

DATA ABSTRACTION IN DBMS

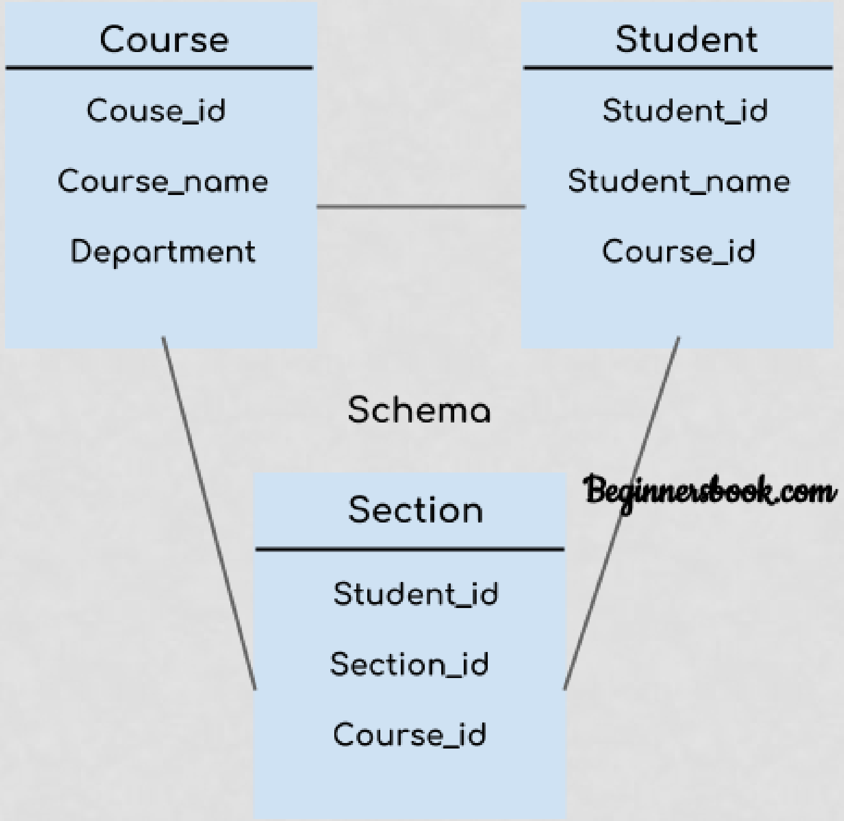


Three Levels of data abstraction

- **Physical level**: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.
- **Logical level**: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.
- **View level**: Highest level of data abstraction. This level

INSTANCE AND SCHEMA IN DBMS

- **DBMS Schema**
- **Definition of schema**: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.



- Physical SchemaThe design of a database at physical level is called **physical schema**, how the data stored in blocks of

SCHEMAS IN DBMS

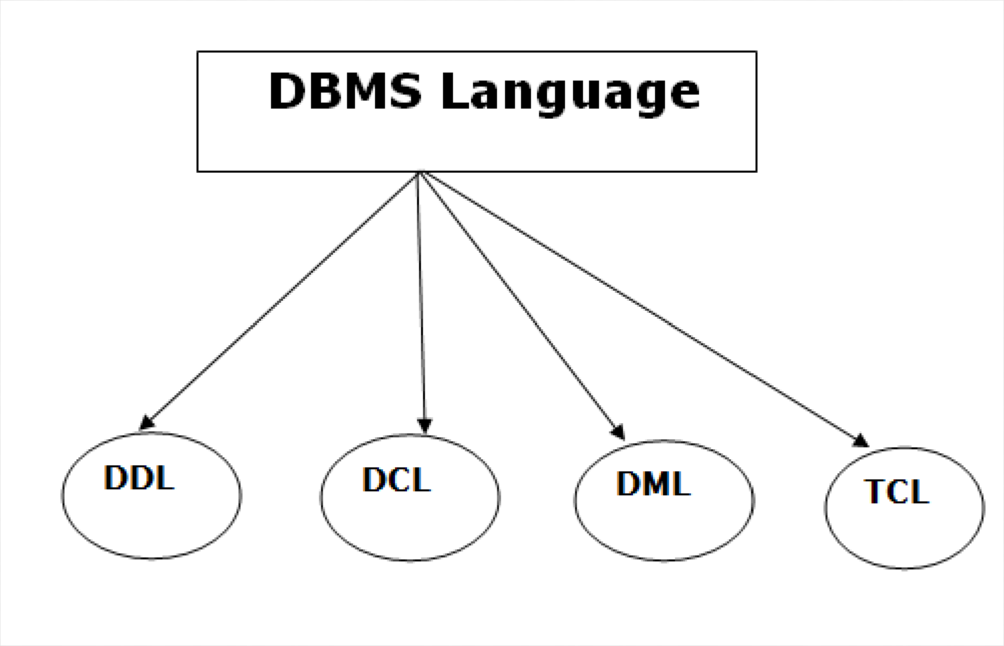
- Logical Schema:
- Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data

- View Schema:
- Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

DBMS INSTANCE

- **Definition of instance**: The data stored in database at a particular moment of time is called instance of database.
- Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

DATABASE LANGUAGES



DATA DEFINITION LANGUAGE

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.
- Here are some tasks that come under DDL:
- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.

DATA MANIPULATION LANGUAGE

- **DML** stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.
- Here are some tasks that come under DML:
- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.

DATA CONTROL LANGUAGE

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.
- (But in Oracle database, the execution of data control language does not have the feature of rolling back.)
- Here are some tasks that come under DCL:
- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.
- There are the following operations which have the authorization of Revoke:
- CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

TRANSACTION CONTROL LANGUAGE

- TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.
- Here are some tasks that come under TCL:
- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

FILE PROCESSING SYSTEM

- **A file processing system helps people keep track of files as they move throughout the various departments of a business.**
- It should be recorded in such a way that
- Should be able to get the data any point in time latter
- Should be able to add details to it whenever required
- Should be able to modify stored information, as needed
- Should also be able to delete them

DISADVANTAGES OF FILE PROCESSING SYSTEM

- Data Redundancy
- Data Inconsistency
- Difficult in Accessing Data
- Integrity Problem
- Data Isolation
- Concurrent Access Problem
- Atomicity Problem

FILE SYSTEM VERSUS DBMS

FILE SYSTEM	DBMS
Software that manages the data files in a computer system	Software to create and manage databases
Helps to store a collection of raw data files into the hard disk	Helps to easily store, retrieve and manipulate data in a database
Tasks such as storing, retrieving and searching are done manually, so it is difficult to manage data	Operations such as updating, searching, selecting data is easier since it allows using SQL querying
Has data inconsistency	Provides higher data consistency using normalization
There is more redundant data	There is low data redundancy
Provides more security to data	Comparatively less data security

TRANSACTION MANAGEMENT

- A transaction is a logical unit of work that contains one or more SQL statements
- It is a collection of operations that performs a single logical function in a database application
- A transaction is an atomic unit
- Transactions in a database environment have two main purposes:
- Reliability of data
- Concurrent Access of Data

DATA MODELS

- Data models can facilitate interaction among the designer, the application programmer and the end user.
- A well- developed data model can even foster improved understanding of the organization for which the database design is developed.
- Data models are a communication tool.

IMPORTANCE OF DATA MODELS

- **The data model is the blueprint.**
- **The data model is the requirements.**
- **The data model is the specifications.**
- **The data model is reusable.**

THE BASIC BUILDING BLOCKS OF DATA MODELS

- The basic building blocks of all data models are entities, attributes, and relationships.
- An entity is anything, such as a person, place, thing, or event, about which data are to be collected and stored. Entities may be physical objects such as customers or products. But entities may also be abstractions such as flight routes or musical concerts.
- An attribute is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit. The attributes are

TYPES OF RELATIONS

- One-to-many (1:M) relationship. A painter paints many different paintings, but each one of them is painted by only one painter. Thus the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER paints PAINTING" as 1:M. Similarly, a customer (the "one") might generate many invoices, but each invoice (the "many") is generated by only a single customer. The "CUSTOMER generates INVOICE" relationship would also be labeled 1:M.
- Many-to-many (M:N or M:M) relationship. An employee might learn many job skills, and each job skill might be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many classes, and each class can be taken by many students, thus yielding the M:N relationship label for the relationship expressed by "STUDENT

DATABASE MANAGEMENT SYSTEM

UNIT 2

KEYS

- **Candidate Key -** The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.
- **Primary Key -** The primary key is selected from one of the candidate keys and becomes the identifying key of a table. It can uniquely identify any data row of the table.
- **Super Key -** Super Key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in the table.
- **Composite Key -** If any single attribute of a table is not capable of being the key i.e it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.
- **Secondary Key -** Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.
- **Foreign Key -** A foreign key is an attribute value in a table that acts as the primary key in another another. Hence, the foreign key is useful in linking together two tables. Data should be entered in the foreign key column with great care, as wrongly entered data can invalidate the relationship between the two tables.

NORMALIZATION

- If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.
- **Update anomalies** − If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies** − We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies** − We tried to insert data in a record that does not exist at all.

FIRST NORMAL FORM

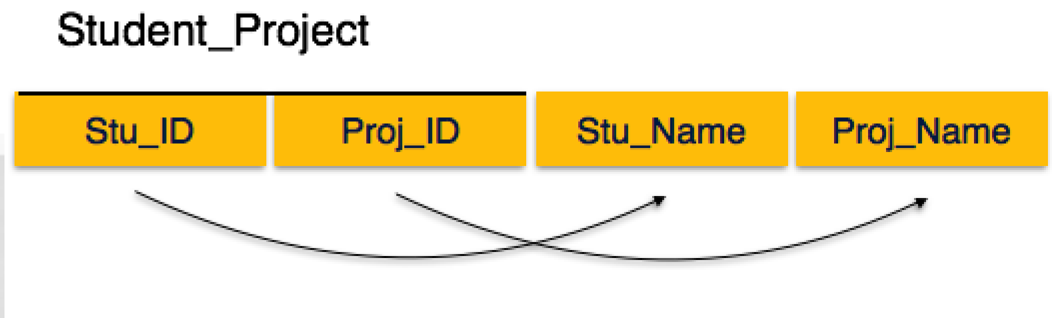
- First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

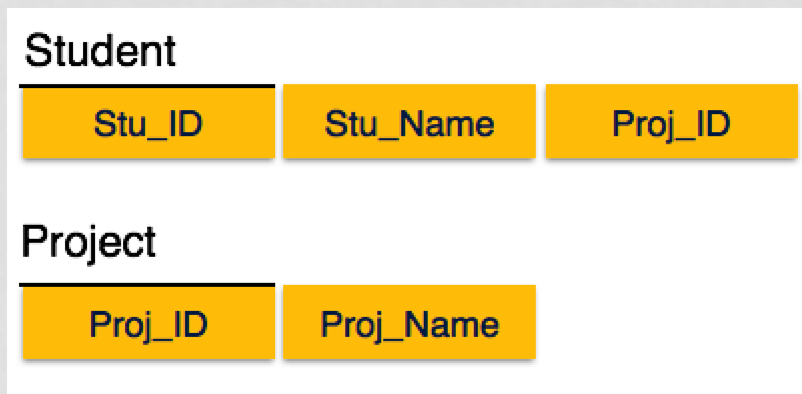
Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

SECOND NORMAL FORM

- Before we learn about the second normal form, we need to understand the following −
- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
- If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X, for which $Y \rightarrow A$ also holds true.



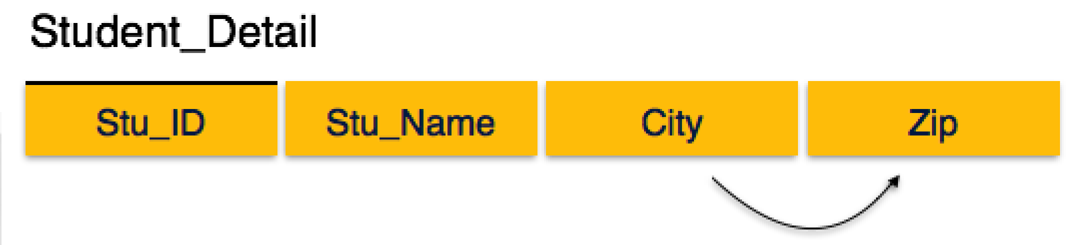
We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.



So there exists no partial dependency.

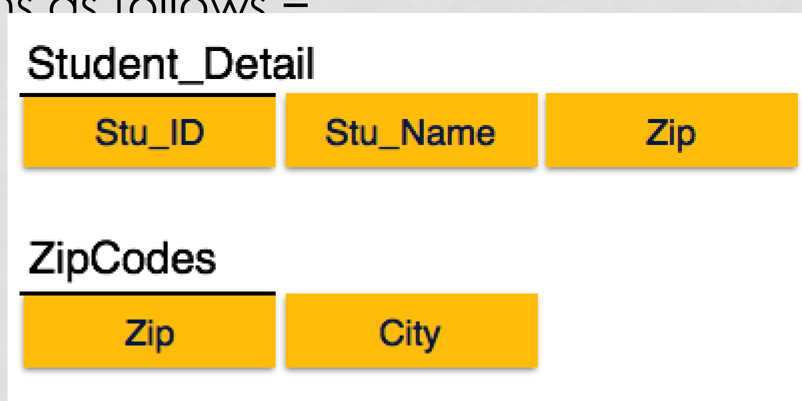
THIRD NORMAL FORM

- For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy –
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, $X \rightarrow A$, then either –
  - X is a superkey or,
  - A is prime attribute.



We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows –



BOYCE-CODD NORMAL FORM

- Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that −
- For any non-trivial functional dependency, X → A, X must be a super-key.
- In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,
- Stu_ID → Stu_Name, Zip
- and
- Zip → City
- Which confirms that both the relations are in BCNF.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF/BCNF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

COLUMN ATTRIBUTE

- <u>UNSIGNED</u>
- Unsigned allows us to enter positive value; you cannot give any negative number
- <u>NOT NULL</u>
- Means column can not be empty
- <u>DEFAULT</u>
- It is used to give a column a fixed value.
- <u>AUTO_INCREMENT</u>
- Auto-increment allows_a unique number to be generated automatically when a new record is inserted into a table.
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

ALTER

```
alter table fyit
add fee decimal(4,2) unsigned default 2050;

describe fyit;

alter table fyit
add fee decimal(4,2) unsigned default 2050 after Std_name;
```

CHANGE
(CHANGING COLUMN NAME)

Alter table fyit

change fee Tution_Fee decimal(4,2) unsigned default 2050;

DELETING COLUMN

Alter table fyit
drop Tution_Fee decimal(4,2) unsigned default 2050;

DELETE/ADD PRIMARY KEY

desc fyit;

alter table fyit
Drop primary key;

alter table fyit
add primary key(std_id);

RENAMING TABLE

alter table fyit rename fybscit;

Or

Rename table fyit to fybscit;

BUILT IN FUNCTIONS(STRING)

```
mysql> SELECT LOWER('DATABASE');
+-------------------+
| LOWER('DATABASE') |
+-------------------+
| database          |
+-------------------+
1 row in set (0.03 sec)

mysql> SELECT UPPER('management');
+---------------------+
| UPPER('management') |
+---------------------+
| MANAGEMENT          |
+---------------------+
1 row in set (0.00 sec)

mysql> SELECT REVERSE('Mumbai');
+-------------------+
| REVERSE('Mumbai') |
+-------------------+
| iabmuM            |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT LENGTH('wELCOME ALL');
+-----------------------+
| LENGTH('wELCOME ALL') |
+-----------------------+
|                    11 |
+-----------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT LENGTH('wELCOME ALL');
+----------------------+
| LENGTH('wELCOME ALL') |
+----------------------+
|                   11 |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT LEFT('Mumbai',2);
+------------------+
| LEFT('Mumbai',2) |
+------------------+
| Mu               |
+------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT RIGHT('Mumbai',2);
+-------------------+
| RIGHT('Mumbai',2) |
+-------------------+
| ai                |
+-------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> SELECT MID('Mumbai',3,1);
+-------------------+
| MID('Mumbai',3,1) |
+-------------------+
| m                 |
+-------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT MID('Mumbai',3);
+-----------------+
| MID('Mumbai',3) |
+-----------------+
| mbai            |
```

```
mysql> SELECT CONCAT('HELLO' 'Mumbai');
+-------------------------+
| CONCAT('HELLO' 'Mumbai') |
+-------------------------+
| HELLOMumbai             |
+-------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT('HELLO', NULL, 'Mumbai');
+---------------------------------+
| CONCAT('HELLO', NULL, 'Mumbai') |
+---------------------------------+
| NULL                            |
+---------------------------------+
1 row in set (0.00 sec)
```

BUILT IN FUNCTIONS(DATE FUNCTIONS)

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2019-07-30 11:12:39 |
+---------------------+
1 row in set (0.03 sec)

mysql> SELECT TIME('2019-07-30 11:12:39');
+-----------------------------+
| TIME('2019-07-30 11:12:39') |
+-----------------------------+
| 11:12:39                    |
+-----------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE('2019-07-30 11:12:39');
+-----------------------------+
| DATE('2019-07-30 11:12:39') |
+-----------------------------+
| 2019-07-30                  |
+-----------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2019-07-30 |
+------------+
1 row in set (0.00 sec)

mysql> SELECT DAY('2019-07-30');
+-------------------+
| DAY('2019-07-30') |
+-------------------+
|                30 |
+-------------------+
1 row in set (0.02 sec)

mysql> SELECT MONTH('2019-07-30');
+---------------------+
| MONTH('2019-07-30') |
+---------------------+
|                   7 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT YEAR('2019-07-30');
+--------------------+
| YEAR('2019-07-30') |
+--------------------+
|               2019 |
+--------------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2019-07-30');
+-----------------------+
| DAYNAME('2019-07-30') |
+-----------------------+
| Tuesday               |
+-----------------------+
1 row in set (0.00 sec)

mysql> SELECT MONTHNAME('2019-07-30');
+-------------------------+
| MONTHNAME('2019-07-30') |
+-------------------------+
| July                    |
+-------------------------+
1 row in set (0.00 sec)
```

NUMERICAL FUNCTION

```
mysql> SELECT ABS(-5);
+---------+
| ABS(-5) |
+---------+
|       5 |
+---------+
1 row in set (0.00 sec)

mysql> SELECT ABS(3);
+--------+
| ABS(3) |
+--------+
|      3 |
+--------+
1 row in set (0.00 sec)
```

```
mysql> SELECT POW(2,3);
+----------+
| POW(2,3) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT MOD(10,3);
+-----------+
| MOD(10,3) |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT MOD(-10,3);
+------------+
| MOD(-10,3) |
+------------+
|         -1 |
+------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(3.78654);
+----------------+
| ROUND(3.78654) |
+----------------+
|              4 |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT SQRT(144);
+-----------+
| SQRT(144) |
+-----------+
|        12 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT ROUND(3.78654,2);
+------------------+
| ROUND(3.78654,2) |
+------------------+
|             3.79 |
+------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT FLOOR(-14.87);
+---------------+
| FLOOR(-14.87) |
+---------------+
|           -15 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT CEIL(-14.87);
+--------------+
| CEIL(-14.87) |
+--------------+
|          -14 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> use university;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_university |
+---------------------+
| dept                |
| pay                 |
+---------------------+
2 rows in set (0.00 sec)

mysql> desc pay;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| empid  | smallint(6)  | NO   | PRI |         |       |
| name   | varchar(20)  | YES  |     | NULL    |       |
| salary | decimal(7,2) | YES  |     | NULL    |       |
| dob    | date         | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

```
mysql> select * from pay;
+-------+-------+----------+------------+
| empid | name  | salary   | dob        |
+-------+-------+----------+------------+
|     1 | Raj   | 45000.00 | 1979-06-18 |
|     2 | Hetal | 42000.00 | 1969-06-12 |
|     3 | Siya  | 40000.00 | 1980-12-12 |
+-------+-------+----------+------------+
3 rows in set (0.00 sec)
```

```
mysql> select dayname(DOB)
    -> from pay
    -> where empid=1;
+--------------+
| dayname(DOB) |
+--------------+
| Monday       |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> select name, salary, salary*.10 from pay;
+-------+----------+------------+
| name  | salary   | salary*.10 |
+-------+----------+------------+
| Raj   | 45000.00 |  4500.0000 |
| Hetal | 42000.00 |  4200.0000 |
| Siya  | 40000.00 |  4000.0000 |
+-------+----------+------------+
3 rows in set (0.02 sec)
```

RELATIONAL ALGEBRA

- Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.
- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output.
- It uses operators to perform queries. An operator can be either **unary** or **binary**.
- They accept relations as their input and yield relations as their output.
- Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

THE FUNDAMENTAL OPERATIONS OF RELATIONAL ALGEBRA ARE AS FOLLOWS –

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

SELECT OPERATION (Σ)

- It selects tuples that satisfy the given predicate from a relation.
- **Notation** − $\sigma_p(r)$
- Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and**, **or**, and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

EXAMPLE

Example :

R

(A B C)

1	2	4
2	2	3
3	2	3
4	3	4

- π (σ (c>3)R) will show following tuples.

- A B C
- -------
- 1 2 4
- 4 3 4

PROJECT OPERATION (Π)

- Projection is used to project required column data from a relation.
- It projects column(s) that satisfy a given predicate.
- Notation – $\Pi_{A1, A2, An}$ (r)
- Where A_1, A_2 , A_n are attribute names of relation **r**.
- Duplicate rows are automatically eliminated, as relation is a set.

EXAMPLE

R

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

π (BC)

B	C
2	4
2	3
3	4

UNION OPERATION

- **Notation** − r U s
- Where **r** and **s** are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold −
- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

EXAMPLE

$$\prod_{author} (Books) \cup \prod_{author} (Articles)$$

Output – Projects the names of the authors who have either written a book or an article or both.

SET DIFFERENCE (−)

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- **Notation** – **r − s**
- Finds all the tuples that are present in **r** but not in **s**.

EXAMPLE

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output – Provides the name of authors who have written books but not articles.

CARTESIAN PRODUCT (X)

- Combines information of two different relations into one.
- **Notation** – r X s
- Where **r** and **s** are relations and their output will be defined as –
- r X s = { q t | q ∈ r and t ∈ s}

$$\sigma_{author\ =\ 'Raheja'}(Books\ X\ Articles)$$

Output – Yields a relation, which shows all the books and articles written by Raheja.

RENAME OPERATION (Ρ)

- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.
- **Notation** – ρ_{x} (E)
- Where the result of expression **E** is saved with name of **x**.

UPDATE AND SET

```
mysql> select * from pay;
+-------+------+----------+------------+
| empid | name | salary   | dob        |
+-------+------+----------+------------+
|     1 | Raj  | 45000.00 | 1979-06-18 |
|     2 | Riya | 42000.00 | 1969-06-12 |
|     3 | Siya | 40000.00 | 1980-12-12 |
+-------+------+----------+------------+
3 rows in set (0.02 sec)

mysql> update pay
    -> set name='Hetal'
    -> where empid=2;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from pay;
+-------+-------+----------+------------+
| empid | name  | salary   | dob        |
+-------+-------+----------+------------+
|     1 | Raj   | 45000.00 | 1979-06-18 |
|     2 | Hetal | 42000.00 | 1969-06-12 |
|     3 | Siya  | 40000.00 | 1980-12-12 |
+-------+-------+----------+------------+
3 rows in set (0.00 sec)
```

```
mysql> create table Dept
    -> (empid smallint unsigned primary key,
    -> empname varchar(20),
    -> deptid smallint,
    -> city varchar(20),
    -> Salary decimal(7,2),
    -> DOJ date);
Query OK, 0 rows affected (0.09 sec)
```

- Update Dept
- Set salary = salary+1000
- Where city ="Mumbai";

JOIN

- A relational database consists of multiple related tables linking together using common columns which are known as foreign key columns. Because of this, data in each table is incomplete from the business perspective.
- MySQL supports the following types of joins:
- Cross join
- Inner join
- Left join
- Right join

CROSS JOIN(CREATE TABLE 1ST)

- CREATE TABLE t1 (
-     id INT PRIMARY KEY,
-     pattern VARCHAR(50) NOT NULL
-);
-
- CREATE TABLE t2 (
-     id VARCHAR(50) PRIMARY KEY,
-     pattern VARCHAR(50) NOT NULL
-);

INSERTING VALUES

```
INSERT INTO t1(id, pattern)
VALUES(1,'Divot'),
      (2,'Brick'),
      (3,'Grid');

INSERT INTO t2(id, pattern)
VALUES('A','Brick'),
      ('B','Grid'),
      ('C','Diamond');
```

USE OF CROSS JOIN

The CROSS JOIN makes a Cartesian product of rows from multiple tables. Suppose, you join t1 and t2 tables using the CROSS JOIN, the result set will include the combinations of rows from the t1 table with the rows in the t2 table.

```
SELECT
    t1.id, t2.id
FROM
    t1
CROSS JOIN t2;
```

id	id
1	C
1	B
1	A
2	C
2	B
2	A
3	C
3	B
3	A

INNER JOIN

- To form an INNER JOIN, you need a condition which is known as a join-predicate.
- An INNER JOIN requires rows in the two joined tables to have matching column values.
- The INNER JOIN creates the result set by combining column values of two joined tables based on the join-predicate.
- To join two tables, the INNER JOIN compares each row in the first table with each row in the second table to find pairs of rows that satisfy the join-predicate.
- Whenever the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of the two tables are included in the result set.

INNER JOIN

- SELECT
- t1.id, t2.id
- FROM
- t1
- INNER JOIN
- t2 ON t1.pattern = t2.pattern;

id	id
2	A
3	B

LEFT JOIN

- Similar to an INNER JOIN, a LEFT JOIN also requires a join-predicate.
- When joining two tables using a LEFT JOIN, the concepts of left table and right table are introduced.
- Unlike an INNER JOIN, a LEFT JOIN returns all rows in the left table including rows that satisfy join-predicate and rows that do not.
- For the rows that do not match the join-predicate, NULLs appear in the columns of the right table in the result set.

LEFT JOIN

- SELECT
-     t1.id, t2.id
- FROM
-     t1
-         LEFT JOIN
-     t2 ON t1.pattern = t2.pattern
- ORDER BY t1.id;

id	id
1	NULL
2	A
3	B

RIGHT JOIN

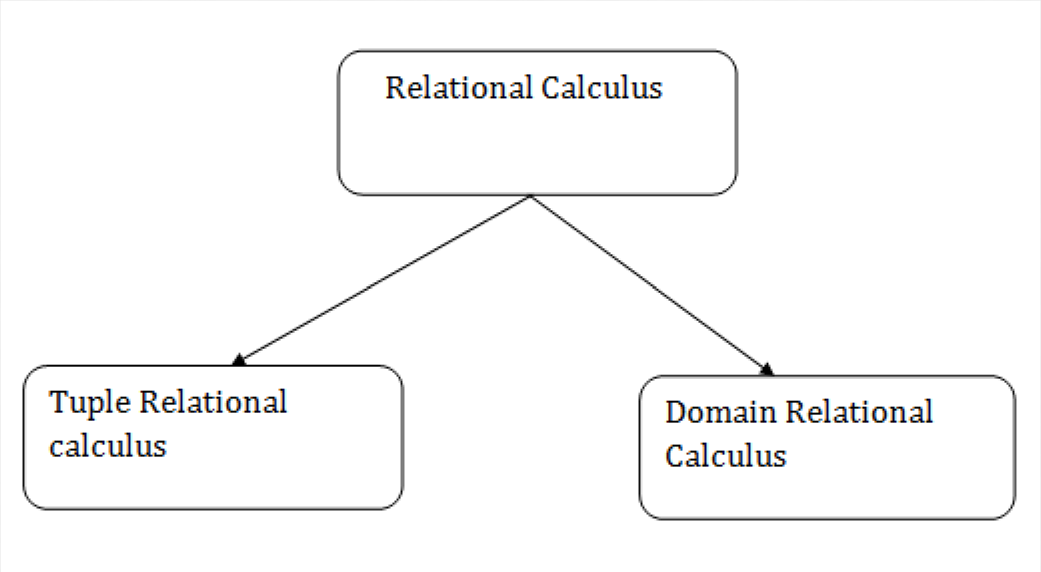
- A RIGHT JOIN is similar to the LEFT JOIN except that the treatment of tables is reversed.
- With a RIGHT JOIN, every row from the right table (t2) will appear in the result set.
- For the rows in the right table that do not have the matching rows in the left table (t1), NULLs appear for columns in the left table (t1).

RIGHT JOIN

- SELECT
- t1.id, t2.id
- FROM
- t1
- RIGHT JOIN
- t2 on t1.pattern = t2.patte
- ORDER BY t2.id;

	id	id
▶	2	A
	3	B
	NULL	C

RELATIONAL CALCULAS



TUPLE RELATIONAL CALCULUS

- In this form of relational calculus, we define a tuple variable, specify the table(relation) name in which the tuple is to be searched for, along with a condition.
- We can also specify column name using a . dot operator, with the tuple variable to only get a certain attribute(column) in result.
- A lot of information, right! Give it some time to sink in.
- A tuple variable is nothing but a name, can be anything, generally we use a single alphabet for this, so let's say T is a tuple variable.

DOMAIN RELATIONAL CALCULUS (DRC)

- In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.
- Syntax: { c1, c2, c3, ..., cn | F(c1, c2, c3, ... ,cn)}
- where, c1, c2... etc represents domain of attributes(columns) and F defines the formula including the condition for fetching the data.

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

BASIS FOR COMPARISON	RELATIONAL ALGEBRA	RELATIONAL CALCULUS
Basic	Relational Algebra is a Procedural language.	Relational Claculus is Declarative language.
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we have to obtain.
Order	Relational Algebra describes the order in which operations have to be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Claculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.